Attorney's Docket No.: 10559-595001 / P21147

APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: GENERATING A LOGIC DESIGN

APPLICANT: WILLIAM R. WHEELER AND MATTHEW J. ADILETTA

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No. EL 624 273 708 US

I hereby certify that this correspondence is being deposited with the United States Postal Service as Express Mail Post Office to Addressee with sufficient postage on the date indicated below and is addressed to the Commissioner for Patents, Washington, D.C. 20231.

August 29, 2001
Date of Deposit

Samantha Bell
Signature

Samantha Bell
Typed or Printed Name of Person Signing Certificate

GENERATING A LOGIC DESIGN

TECHNICAL FIELD

This invention relates to integrated circuit design tools.

BACKGROUND

Logic designs for integrated circuits (IC) typically include either schematic design or text design. A schematic design shows a computer chip with logic elements as a two-dimension diagram. Logic elements are either state elements (e.g., flip-flops, latches, etc.) or combinatorial elements (e.g. AND gates, NOR gates, etc.). Various geometric figures represent the logic elements. Lines drawn into or out of the logic elements generally represent input, output, clock, or enabling signals. Lines connecting such geometric shapes indicate a functional logic relationship between the logic elements.

A textual representation describes the logic elements of the computer chip using one-dimensional text lines. Textual representations are used in hardware description languages (HDLs) which allow designers to simulate logic designs prior to forming the logic on silicon. Examples of such languages include Verilog and Very High-Level Design Language (VHDL). Using these languages, a designer can write code to simulate a logic design and execute the code in order to determine if the logic design performs properly.

Standard computer languages may also be used to simulate a logic design. One example of a standard computer language that may be used is C++.

DESCRIPTION OF THE DRAWINGS

FIG. 1 is a flowchart showing a process for generating a logic design.

FIG. 2 is a block diagram of a computer system on which the process of FIG. 1 may be performed.

DESCRIPTION

Referring to FIG. 1, a process 10 is shown for generating a logic design which includes both schematic design and textual design features. Process 10 may be implemented using a computer program running on a computer or other type of machine, as described in more detail below. The process 10 accesses a unified database that represents a complete visual model of an integrated circuit (IC) and which embeds a combinatorial one-dimensional data block. The combinatorial data block allows for a blending of the textual design into a two-dimensional representation. In addition, a set of abstractions is used by process 10 to shorten the development time of the unified database. The unified database can be used by both designers and implementers of IC design to create an IC without referring to two separate and disconnected design schemes.

The unified database is represented in Register Transfer Diagrams (RTDs), which are two-dimensional representations of the IC design. RTDs are hierarchical diagrams that illustrate all state elements of an IC design while allowing a user to abstract the combinatorial logic into simple boxes. RTDs concisely convey (1) all state elements, (2) the partitioning of work to be accomplished between the state elements, (3) the flow of data through state elements, (4) the partitioning and logical organization of the design within and between levels of hierarchy, (5) the intent of the design and (6) signal information.

In operation, process 10 generates the combinatorial data block (12). The IC designer determines that in a particular portion of the IC design a combinatorial element is required. A textual description is used, to represent the combinatorial element. The textual description is preferable in a simplified form to avoid complexities introduced by complex textual descriptions that otherwise need to be accounted for in the design process. Thus, the combinatorial data block includes a textual description that is in a simplified form in order to ease integration into two-dimensional or graphics level scheme. The simplification follows a set of design capture rules. For an example, a combinatorial data block is represented in Verilog as follows:

```
always @()
begin
   case (f2_ctx_w)   // synopsys    parallel_case
          3'b000: next_seq_thd_w = 3'b001;
          3'b001: next_seq_thd_w = 3'b010;
          3'b010: next_seq_thd_w = 3'b011;
          3'b011: next_seq_thd_w = 3'b100;
          3'b100: next_seq_thd_w = 3'b101;
          3'b101: next_seq_thd_w = 3'b110;
          3'b110: next_seq_thd_w = 3'b111;
          3'b111: next_seq_thd_w = 3'b000;
     endcase
end
```

The design capture rules used to simplify the combinatorial data block in this example include: (1) avoiding the use of declarations and (2) avoiding entries in a sensitivity list. If declarations and entries were used, these fields would need to be changed if the IC design changes and the combinatorial block were affected. By not allowing declarations or entries in the sensitivity list in the combinatorial data block, process 10 eliminates the need for the IC designer to update these fields during the IC development process. In other words, as changes occur in the IC design, there are no manual updates needed by the IC designer to account for these changes. By setting-up these restrictions, there is less opportunity for human error when design changes occur. Other design capture rules may be implemented to simplify the combinatorial data block and eliminate unnecessary updates as the design develops.

Process 10 imports the combinatorial data block (14). In this embodiment, this is performed on a computer system as described below through an input/output interface (e.g., mouse, keyboard). When the combinatorial data block is imported to the logic design system, process 10 checks to ensure that the design capture rules for generating the combinatorial data block were followed from 12 (16). Process 10 notifies the designer if an error has occurred (18). For example, an error message is displayed on the IC designer's computer screen.

Process 10 uses a set of abstractions to facilitate the development of the unified database (20). The set of abstractions are abbreviated representations of various logic components. For example, a comparator has thousands of transistors. The creation of each and every transistor in the IC design or carrying the data associated with each transistor would be cumbersome. The abbreviated representation would be a block diagram with an input and an output. Abstractions can be instantiated from a library so that creating a logical component from an abstraction is fast and easy for a designer to do. For example, the IC design tools employing process 10 reside on a personal computer and the tools operate in a MS-Windows® environment. If the IC designer determines a comparator is needed in the design, the designer pulls-down a menu in the application and selects a comparator. Subsequent boxes appear and the designer checks-off blocks as to the parameters (e.g., inputs) needed for the comparator. After the designer chooses the logic component by using the set of abstractions, it is saved in the unified database.

Process 10 embeds the combinatorial data block into the two-dimensional schematic presentation to complete the unified data base (22) Thus, the unified data base is a complete representation of the IC and can be represented in RTDs.

Normally, during an IC design process, designers implement block diagrams at the start of the design process and develop the design using RTL code, a one-dimensional text description. Often the block diagrams are not kept up-to-date because the designer makes all the changes to the RTL so that the RTL becomes the design code. The unified database generated by process 10 ensures configuration management of the IC design by keeping all the design information in one location throughout the design process. Thus, this logic design scheme eliminates traditional ambiguities that occur between previous implementation and design models because of the constant iterations of reconciling both the schematic and textual design schemes. Having a unified database, process 10 allows for the generation of C++ and Verilog from one location. It also allows for generation of synthesizable Verilog from textual and visual elements.

FIG. 2 shows a computer 40 for generating a logic design using process 10. Computer 40 includes a processor 42, a memory 44, and a storage medium 46 (e.g., a hard disk). Storage medium 46 stores data 52 which defines a logic design, a graphics library 50 for implementing the logic design, and machine-executable instructions 48, which are executed by processor 42 out of memory 44 to perform process 10 on data 52.

Process 10, however, is not limited to use with the hardware and software of FIG. 2; it may find applicability in any computing or processing environment. Process 10 may be implemented in hardware, software, or a combination of the two. Process 10 may be implemented in computer programs executing on programmable computers or other machines that each includes a processor, a storage medium readable by the processor (including volatile and non-volatile memory and/or storage elements), at least one input device, and one or more output devices. Program code may be applied to data entered using an input device, such as a mouse or a keyboard, to perform process 10 and to generate a simulation.

Each such program may be implemented in a high level procedural or object-oriented programming language to communicate with a computer system. However, the programs can be implemented in assembly or machine language. The language may be a compiled or an interpreted language.

Each computer program may be stored on an article of manufacture, such as a storage medium or device (e.g., CD-ROM, hard disk, or magnetic diskette), that is readable by a general or special purpose programmable machine for configuring and operating the machine when the storage medium or device is read by the machine to perform process 10. Process 10 may also be implemented as a machine-readable storage medium, configured with a computer program, where, upon execution, instructions in the computer program cause the machine to operate in accordance with process 10.

The invention is not limited to the specific embodiments set forth above. For example, process 10 is not limited to embedding one-dimensional design into a two-dimensional design. Process can be any n-dimensional design embedded into a (n+m)-dimensional design, where $n \geq 1$ and $m \geq 1$. Process 10 is not limited to the computer languages set forth above, e.g., Verilog, C++, and VHDL. It may be implemented using any appropriate computer language. Process 10 is also not limited to the order set forth in FIG. 1. That is, the blocks of process 10 may be executed in a different order than that shown to produce an acceptable result.

Other embodiments not described herein are also within the scope of the following claims.

What is claimed is: